Optimization of Model Load Time using Model Caching Capability in Intel® Distribution of OpenVINO™ Toolkit

White Paper

March 2023

Author:
Ramesh Perumal

Document number: 774938-1.0

Contents

Tables

Figures

Revision History

Date	Revision	Description
March 2023	1.0	Initial release

§

1.0 Introduction

This document presents the BKM for reducing the mode load time by enabling model caching in Intel® Distribution of OpenVINO™ Toolkit. The following figure illustrates the steps involved in implementing the inference pipeline in the user application using OpenVINO™ Runtime API.

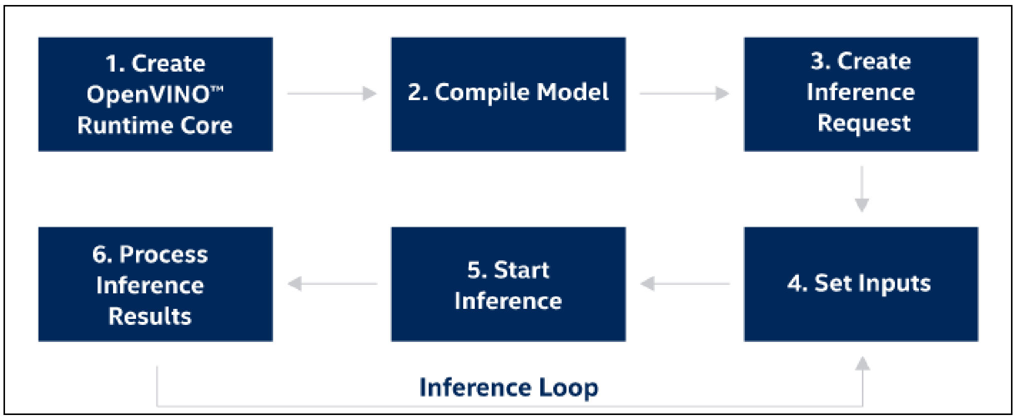

1. Create OpenVINO™ Runtime Core
2. Compile Model
3. Create Inference Request
4. Set Inputs
5. Start Inference
6. Process Inference Results
Inference Loop

Figure 1. Workflow for Integrating OpenVINO™ Runtime API with User Application

Step 2 in Figure 1 involves loading and compiling the model by performing several time-consuming device-specific optimizations and network compilations. To avoid this recurring delay on application startup, this white paper illustrates the method to enable model caching capability in OpenVINO™ and demonstrates a significant reduction in model load time on CPU and GPU in 11th Gen Intel® Core™ i5-1145G7.

1.1 Acronyms

Table 1. Acronyms

Term	Description
BKM	Best Known Method
OpenVINO™	Open Visual Inference & Neural Network Optimization
DUT	Device Under Test

1.2 Reference Documents

Log in to the Resource and Documentation Center (rdc.intel.com) to search and download the document numbers listed in the following table. Contact your Intel field representative for access.

Note: Third-party links are provided as a reference only. Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether the referenced data is accurate.

Table 2. Reference Document

Document	Document No./Location
OpenVINO™ Toolkit	https://software.seek.intel.com/openvino-toolkit
OpenVINO™ Docker Image	https://hub.docker.com/r/openvino/ubuntu20_dev
Integrating OpenVINO™ Runtime API with user application	https://docs.openvino.ai/latest/openvino_docs_OV_UG_Integrate_OV_with_your_application.html#doxid-openvino-docs-o-v-u-g-integrate-o-v-with-your-application
Model Caching	https://docs.openvino.ai/latest/openvino_docs_OV_UG_Model_caching_overview.html

§

2.0 Model Caching

2.1 Enabling Model Caching in OpenVINO™ Core

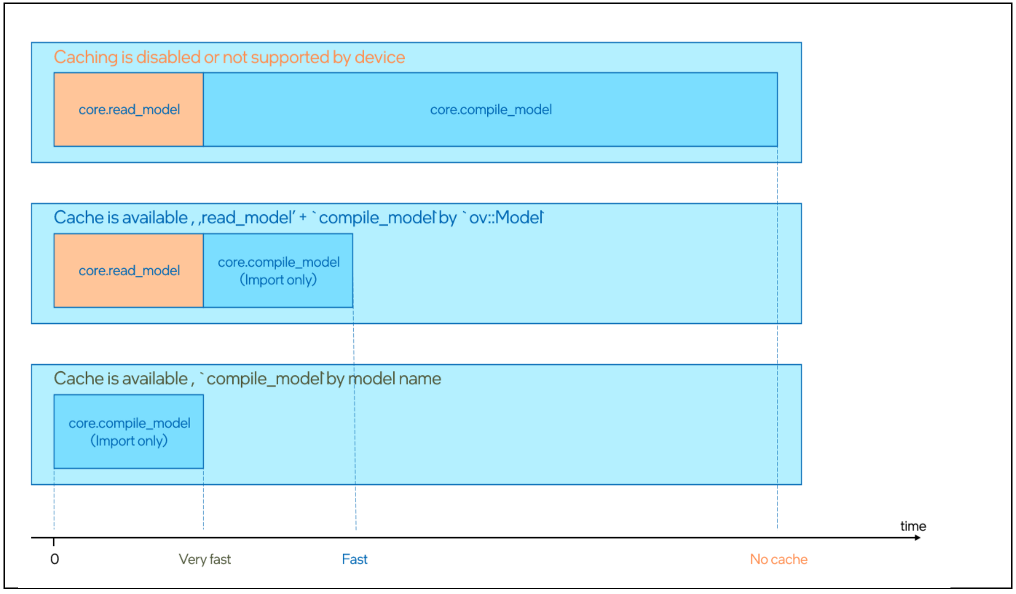
The model caching is enabled by setting 'CACHE_DIR' property in the OpenVINO™ Core as shown below:

```
from openvino.runtime import Core
# 1. Create OpenVINO Core object
core = Core()
# 2. Enable model caching
core.set_property({'CACHE_DIR': '/path/to/cache/dir'})
# 3. Read model
model = core.read_model(model=xml_path)
# 4. Compile model
compiled_model = core.compile_model(model=model,
device_name=device_name)
```

With this code, if the device specified by `device_name` supports import/export model capability, a cached blob is automatically created inside the `/path/to/cache/dir` folder. If the device does not support import/export capability, cache is not created, and no error is thrown.

It is noted that the very first `compile_model` (when cache is not yet created) takes slightly longer time to export the compiled blob into a cache file. However, the succeeding `compile_model` attempts significantly reduce the model load time as it just imports the model from the cached blob. This is clearly shown in Figure 2 by the reduced time taken by the second trace (with model caching) compared to the first trace (without model caching).

Figure 2. Effect of Model Caching on Model Load Time


Caching is disabled or not supported by device
core.read_model
core.compile_model
Cache is available , ,read_model' + `compile_model by `ov::Model
core.read_model
core.compile_model (Import only)
Cache is available , `compile_model by model name
core.compile_model (Import only)
time
0
Very fast
Fast
No cache

If the application does not need to customize the model input and output at every use, then the `compile_model` is implemented in a single call by skipping the `read_model,` follow the steps shown below.

```
from openvino.runtime import Core
# 1. Create OpenVINO Core object
core = Core()
# 2. Enable model caching
core.set_property({'CACHE_DIR': '/path/to/cache/dir'})
# 3. Compile model
compiled_model = core.compile_model(model=xml_path,
device_name=device_name)
```

Consequently, the model load time is further reduced as shown in Figure 2 and the third trace (without `read_model`) is faster than the second trace (with `read_model`).

Table 3. Device Under Test

Hardware	
Model	NUC11TNHv5
CPU	11th Gen Intel® Core™ i5-1145G7 @ 2.60GHz x 8
GPU	Intel® Iris® X^e Graphics
Memory	16 GB
Software	
OS	Ubuntu* 20.04 LTS
Docker	20.10.16
OpenVINO™	2022.3.0

2.2 Performance Evaluation

In this section, the improvement in model load time by enabling model caching is evaluated using a computationally-expensive, Faster-RCNN object detection network (849.9 GFlops, 52.79 MParams) on CPU and GPU in DUT. The steps to reproduce the corresponding results are summarized as follows:

1. Pull the OpenVINO™ docker image and create the container to execute inference on CPU and GPU

```
docker pull openvino/ubuntu20_dev:2022.3.0

docker run -u root -it --name model_cache_ov_2022_3 -v /tmp/.X11-
unix:/tmp/.X11-unix -e DISPLAY="$DISPLAY" -v
/home/ramesh/test:/home/openvino --device /dev/dri:/dev/dri --group-
```

```
add="$(stat -c "%g" /dev/dri/render*)"
openvino/ubuntu20_dev:2022.3.0
```

2. Download the optimized faster-rcnn-resnet101-coco-sparse-60-0001 model

```
omz_downloader --name faster-rcnn-resnet101-coco-sparse-60-0001
```

3. Create a Python* script *eval_load_time.py* with the following source code where the model load time is averaged over 50 iterations by enabling/disabling model caching

```
from openvino.runtime import Core
import time
import numpy as np

core = Core()
# Comment out the following line to disable model caching
core.set_property({'CACHE_DIR':'/home/openvino/model_cache/'})

model_path = '/home/openvino/intel/faster-rcnn-resnet101-coco-
sparse-60-0001/FP16/faster-rcnn-resnet101-coco-sparse-60-
0001.xml'
num_iter = 50
load_time = np.zeros((num_iter,1))
for i in range(num_iter):
    exe_start = time.time()
    # For inferencing with CPU, replace ‘GPU’ with ‘CPU’ in the
    # following line
    compiled_model = core.compile_model(model_path, 'GPU')
    exe_end = time.time()
    load_time[i] = exe_end - exe_start

print("Mean Model Load Time : {} s".format(np.mean(load_time)))
```

4. Run the Python script to get the mean model load time

```
python3 eval_load_time.py
```

2.3 Results

The Python script *eval_load_time.py* is executed on CPU and GPU in DUT to obtain the mean model load time of *faster-rcnn-resnet101-coco-sparse-60-0001* and the corresponding results are presented in Table 4.

Table 4. Model Load Time with/without Model Caching on CPU and GPU in DUT

Inference Platform	Model Load Time* (s)	
	Without Model Caching	With Model Caching
GPU	7.8	1.2
CPU	0.59	0.48

*Model load time is calculated as the mean of its values obtained in 50 iterations

From Table 4, it is evident that the model caching significantly reduces the model load time by 19% and 85% on CPU and GPU, respectively.

§

3.0 Conclusion

The method of enabling model caching in OpenVINO™ Core is presented to reduce the recurring delay in model loading/compiling on application startup. The effect of model caching on the model load time is evaluated with Faster-RCNN on CPU and GPU in 11th Gen Intel® Core™ i5-1145G7 using OpenVINO™ (2022.3.0). The results infer that enabling model caching reduces the model load time by 19% and 85% on CPU and GPU, respectively.

§